Universität Leipzig
Institut für Informatik
Sommersemester 2014

Realisierung von OpenStreetMap-Karten mit leaflet.js und GeoJSON

Belegarbeit im Bachelor-Seminar Informatik,
Sommersemester 2014

25. September 2014

Daniel Obraczka
Arno-Nitzsche-Str. 3
04277 Leipzig
daniel@obraczka.de

Inhaltsverzeichnis

1 Open Street Map

Das freie Projekt OpenStreetMap sammelt für jeden frei nutzbare Geodaten, welche verwendet werden können, um Welt- oder andere Karten zu erstellen. Hierbei wird besonderer Wert darauf gelegt, die Nutzer weder durch restriktive Lizenzen einzuschränken, noch die Nutzung an die Zahlung eines Entgelts zu koppeln, weshalb eine Einbindung in Drucke, Webseiten und Anwendungen wie Navigationssoftware sehr einfach möglich ist. Zur Datennutzung ist hierbei lediglich die Nennung von OpenStreetMap als Datenquelle erforderlich.

OpenStreetMap Karten können auch sehr leicht in Webseiten eingebunden werden. Hierfür ist lediglich ein Text-Editor und etwas Kenntnisse in HTML und CSS nötig. Kenntnisse in Javascript sind zwar nicht zwingend notwendig, sind aber von Vorteil, da im folgenden die JavaScript-Bibliothek leaflet.js genutzt wird.

2 Die JavaScript-Bibliothek leaflet.js

Leaflet bietet sich durch seine Einfachheit, Performance und vielseitige Nützlichkeit an um Karten in Webseiten zu erstellen. Es arbeitet ohne Probleme auf allen großen Desktop- und mobilen Platformen, wobei es die Vorteile von HMTL 5 und CSS 3 moderner Browser nutzt, aber trotzdem auch auf älteren Systemen nutzbar ist.

Leaflet ermöglicht es, mit verschiedenen Layern, Markern, Kreisen, Polylinien, Polygonen und Popups zu arbeiten und so die gesamten Möglichkeiten digitaler Kartendarstellung auszuschöpfen. Im Weiteren werden die wesentlichen Schritte zur Erstellung einer Kartenvisualisierung mit `leaflet.js` erläutert. Dieser Text kann nur als erste Einführung dienen. Für detailliertere Darstellungen zu einzelnen Fragen wird auf die Leaflet-Dokumentation `http://leafletjs.com/reference.html` verwiesen.

2.1 Vorbereiten des HTML-Dokuments

Der erste Schritt beim Erstellen einer Karte mit Leaflet besteht darin sein HTML-Dokument entsprechend vorzubereiten. Um später Kompatibilitätsprobleme zu vermeiden, empfiehlt es sich, die Leaflet-Bibliothek herunterzuladen[1] und lokal in einem Verzeichnis `leaflet` zu entpacken. Die weiteren Ausführungen beziehen sich auf die so installierte Leaflet-Version 0.7.3.

Zunächst muss die Leaflet CSS-Datei im Kopf der HTML-Datei eingebunden werden.

```
1 <link rel="stylesheet" href="leaf"|let/leaf"|let.css" />
```

Ebenso muss muss mit der Leaflet JavaScript-Datei verfahren werden.

```
1 <script src="leaf"|let/leaf"|let.js"></script>
```

In der HTML-Datei muss ein `div` Element mit einer bestimmten `id` an die Stelle gesetzt werden, an der die Karte erscheinen soll.

```
1 <div id="map"></div>
```

Es sollte darauf geachtet werden, dass der `map` Container eine definierte Höhe hat. Diese kann beispielsweise in den `<style>`-Tags gesetzt werden.

[1] Von `http://leafletjs.com/download.html`.

```
<style> #map { width: 800px; height: 500px; } </style>
```

2.2 Erstellen einer Karte

Hat man diese Vorbereitungen getroffen, kann man eine Karte initialisieren und mit ihr arbeiten. Will man beispielsweise eine Karte von Leipzig erstellen, so initialisiert man die `map`, setzt ihren View auf die gewünschten Koordinaten und bestimmt einen Zoom-Level.

```
var map = L.map('map').setView([51.34091, 12.37885], 13);
```

Standardmäßig sind alle Maus- und Touchinteraktionen auf der Karte aktiviert. Da wir beim Erstellen der `map`-Instanz keine Optionen übergeben haben, hat sie Zoom- und Attribution-Controls.

Als nächstes wird eine Kachelebene zur Karte hinzugefügt, welche die Straßen schön darstellt. In diesem Fall wird ein Layer von Mapbox verwendet, da diese nicht nur ästhetisch ansprechend ist, sondern auch die Möglichkeit bietet, eigene Layer zu erstellen und kostenfrei zu nutzen. Um eine Kachelebene zu erstellen wird das URL-Template für das Kachelbild, der Attributionstext und der maximale Zoomlevel des Layers benötigt.

```
L.tileLayer('https://{s}.tiles.mapbox.com/v3/{id}/{z}/{x}/{y}.png', {
  maxZoom: 18,
  attribution: 'Map data &copy; [...]',
  id: 'examples.map-i86knfo3'
}).addTo(map);
```

Die bisherigen Ausführungen führen zu einer Karte, die etwa so aussieht:

2.3 Marker, Kreise und Polygone

Leaflet bietet ebenfalls die Möglichkeit, Marker, Kreise, Polygone und andere geometrische Objekte zur Karte hinzuzufügen. Möchte man beispielsweise einen simplen Marker zur Karte hinzufügen, so erzeugt man diesen einfach an der entsprechenden Stelle.

```
var marker = L.marker([51.34091, 12.37885]).addTo(map);
```

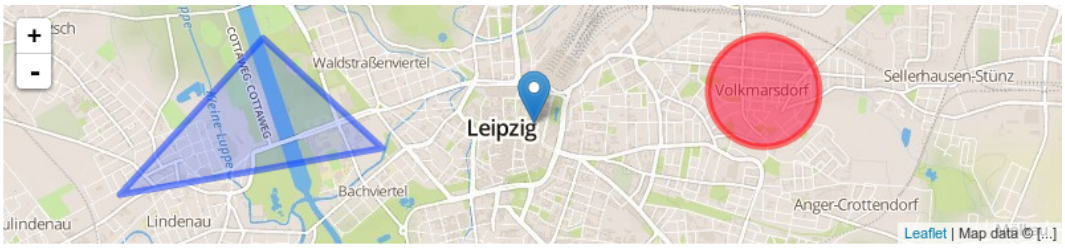
Das Erzeugen eines Kreises verläuft analog, nur muss man hierbei noch den Radius und das Erscheinungsbild des Kreises übergeben.

```
var circle = L.circle([51.34359, 12.40871], 500, {
    color: 'red',
    fillColor: '#f03',
    fillOpacity: 0.5
}).addTo(map);
```

Auch ein Polygon zu erzeugen ist sehr simpel.

```
var polygon = L.polygon([
    [51.33898, 12.35928],
    [51.34788, 12.34383],
    [51.33522, 12.32529]
]).addTo(map);
```

Setzt man die Objekte mit den beschriebenen Eigenschaften in die bereits erzeugte Karte, sieht dies wie folgt aus:



2.4 Arbeiten mit Popups

Popups sind sehr nützlich, um Objekte mit Informationen zu versehen. In Leaflet kann man mit der `bindPopup`-Methode ein Objekt mit einem Popup versehen. Bei Markern gibt es noch die Möglichkeit, mit der Methode `openPopup` das Popup sofort zu öffnen, andere Objekte müssen angeklickt werden.

```
marker.bindPopup("<b>Hallo Welt!</b><br>Ich bin ein Popup").openPopup();
circle.bindPopup("Ich bin ein Kreis");
polygon.bindPopup("Ich bin ein Polygon");
```

Wie man sieht ist es auch möglich den Popuptext mittels HTML zu formatieren. Ebenfalls möglich ist es Popups als Layer zu benutzen. Hierbei wird statt der `addto-` die `openOn`-Methode genutzt, da diese sich darum kümmert bereits geöffnete Popups zu schließen.

```
var popup = L.popup()
                .setLatLng([51.3364, 12.39361])
                .setContent("Ich bin ein alleinstehendes Popup!")
                .openOn(map);
```

2.5 Umgang mit Ereignissen

Jedes Mal, wenn in Leaflet etwas passiert, beispielsweise wenn der Nutzer auf eine Stelle in der Karte klickt, triggert das dazugehörige Element ein Ereignis, welches mit einer Funktion versehen werden kann. Dies ermöglicht eine Vielzahl von Nutzer-Interaktionen. Will man beispielsweise wissen, an welche Stelle man geklickt hat, lässt sich dies einfach mit einem `alert` und `latlng` ausgeben.

```
function onMapClick(e) {
    alert("Es wurde an die Stelle " + e.latlng + " geklickt");
}
map.on('click', onMapClick);
```

Jedes Objekt hat seine eigenen Ereignisse, wie in der Leaflet-Dokumentation genauer beschrieben ist. Das erste Argument der Listener-Funktion ist ein event-Objekt, es enthält Informationen über das Ereignis, welches gerade stattgefunden hat. Im obigen Beispiel hat das `onMapClick` Eventobjekt `e` die Eigenschaft `latlng`, in der die Geokoordinaten von `e` gespeichert sind, womit wir in der Lage sind, diese Koordinaten bei einem Klick auszugeben.

Die Grundtechniken im Umgang mit Leaflet sind damit abgedeckt. Komplexere Datenaggregate werden in Javascript über JSON-Objekte verwaltet. Für geolokale Anwendungen spielt GeoJSON als Standard eine wichtige Rolle. Dieses Format soll nun beschrieben werden.

3 GeoJSON

GeoJSON ist ein Format, das eine breite Palette geographischer Datenstrukturen ermöglicht. Ein GeoJSON-Objekt kann eine geometrische Struktur, ein Feature oder eine Sammlung von Features sein. GeoJSON unterstützt folgende geometrischen Typen: `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon` und `GeometryCollection`. Features enthalten in GeoJSON ein geometrisches Objekt zusammen mit zusätzlichen Eigenschaften. Eine Feature-Sammlung enthält ein Liste von Eigenschaften.

Leaflet unterstützt alle genannten GeoJSON-Typen, aber Features und FeatureCollections sind am gebräuchlisten, da sich dadurch Features mit einer Liste von Eigenschaften erstellen lassen. Ein Beispiel eines simplen GeoJSON-Feature wäre folgendes:

```
var geojsonFeature = {
    "type": "Feature",
    "properties": {
        "name": "Nikolaikirche",
        "amenity": "Kirche",
        "popupContent": "Hier ist die Nikolaikirche"
    },
    "geometry": {
        "type": "Point",
        "coordinates": [12.37856, 51.34033]
    }
};
```

Zu beachten ist hierbei, dass die Koordinaten bei GeoJSON in anderer Reihenfolge übergeben werden als bei Leaflet.

3.1 Erzeugen eines GeoJSON-Layers

GeoJSON-Objekte werden zur Karte mit einem GeoJSON-Layer hinzugefügt. Um einen solchen zu erzeugen und zur Karte hinzuzufügen kann man folgenden Code verwenden:

```
L.geoJson(geojsonFeature).addTo(map);
```

Alternativ ist es auch möglich ein leeres GeoJSON-Layer zu erzeugen und einer Variable zuzuweisen, so dass man diesem später mehr Features hinzufügen kann.

```
var myLayer = L.geoJson().addTo(map);
myLayer.addData(geojsonFeature);
```

3.2 Optionen

Es ist möglich diese Features und das Layer mit verschiedenen Optionen in ihrer Erscheinung zu beeinflussen.

3.2.1 style

So kann man mit der `style`-Option Features in verschiedener Art und Weise darstellen. Zum einen kann man alle Features identisch darstellen.

```
var myLines = [{
    "type": "LineString",
    "coordinates": [[12.36219, 51.33855], [12.38005, 51.33447],
                    [12.42262, 51.34573]] },
  { "type": "LineString",
    "coordinates": [[12.34056, 51.33941], [12.37524, 51.34798],
                    [12.40889, 51.33705]] }
];

var myStyle = {
    "color": "#ff7800",
    "weight": 5,
    "opacity": 0.65
};

L.geoJson(myLines, { style: myStyle }).addTo(map);
```

Alternativ kann eine Funktion übergeben werden, welche je nach Eigenschaft der Features diese unterschiedlich darstellt.

```
var lines = [{
    "type": "Feature",
    "properties": {"farbe": "A"},
    "geometry": {
        "type": "LineString",
        "coordinates": [[12.36219, 51.33855], [12.38005, 51.33447],
                        [12.42262, 51.34573]] }},
  { "type": "Feature",
    "properties": {"farbe": "B"},
    "geometry": {
        "type": "LineString",
        "coordinates": [[12.34056, 51.33941], [12.37524, 51.34798],
                        [12.40889, 51.33705]] }}
];

L.geoJson(lines, {
    style: function(feature) {
        switch (feature.properties.farbe) {
            case 'A': return {color: "#ff0000"};
            case 'B': return {color: "#0000ff"};
        }
    }
}).addTo(map);
```

3.2.2 onEachFeature

Die `onEachFeature`-Option wird für jedes Feature aufgerufen, bevor es zum GeoJSON-Layer hinzugefügt wird. Dies wird oft eingesetzt, um bequem Popups, die beim Anklicken erscheinen, zu Objekten hinzuzufügen.

```
function onEachFeature(feature, layer) {
    // Hat dieses Feature eine Eigenschaft namens "popupContent"?
    if (feature.properties && feature.properties.popupContent) {
        layer.bindPopup(feature.properties.popupContent);
    }
}

var geojsonFeature = {
    "type": "Feature",
    "properties": {
        "name": "Nikolaikirche",
        "amenity": "Kirche",
        "popupContent": "Hier ist die Nikolaikirche"
    },
    "geometry": {
        "type": "Point",
        "coordinates": [12.37856, 51.34033]
    }
};

L.geoJson(geojsonFeature, {
    onEachFeature: onEachFeature
}).addTo(map);
```

3.2.3 Filter

Die `filter`-Option kann genutzt werden, um die Sichtbarkeit von GeoJSON-Objekte zu steuern. Hierzu wird eine Funktion als die `filter`-Option übergeben. Diese Funktion wird für jedes Feature im GeoJSON-Layer aufgerufen und bekommt als Parameter `feature` und `layer` übergeben. Somit kann der Wert in den Eigenschaften des Features genutzt werden, um die Sichtbarkeit zu steuern, indem entweder `true` oder `false` übergeben wird. Beispielsweise ist im folgenden der Punkt Nikolaikirche sichtbar, der Punkt Clara-Zetkin-Park aber nicht.

```
var someFeatures = [{
    "type": "Feature",
    "properties": {
        "name": "Nikolaikirche",
        "show_on_map": true
    },
    "geometry": {
        "type": "Point",
        "coordinates": [12.37856, 51.34033]
    }
}, {
    "type": "Feature",
    "properties": {
        "name": "Clara-Zetkin-Park",
        "show_on_map": false
    },
```

```
    "geometry": {
        "type": "Point",
        "coordinates": [12.35807, 51.33115]
    }
}];

L.geoJson(someFeatures, {
    filter: function(feature, layer) {
        return feature.properties.show_on_map;
    }
}).addTo(map);
```

4 Beschreibung der Realisierung der Karten im Projekt

Um Daten in Karten interaktiv zu visualisieren gibt es verschiedene Möglichkeiten, etwa Pulldown-Menüs, Slider, Radiobuttons, Balkendiagramme, Quadrate, Polygone etc. Im Folgenden wird eine Möglichkeit dargestellt, Stadtteilpolygone nach bestimmten Merkmalsausprägungen einzufärben, wobei zusätzlich für unterschiedliche Jahre die jeweiligen Ausprägungen visualisiert werden. Die dargestellte Implementierung ist sehr einfach und lässt sich sicher weiter verfeinern, aber sie erfüllt ihren Zweck.

Hierbei werden die Daten aus einer vorab erstellten GeoJSON-Datei geladen. Die einzelnen Polygone haben jeweils Eigenschaften, welche von einer Funktion abgefragt werden um das Polygon dementsprechend einzufärben. Zum Ändern des betrachteten Merkmals wird ein Pulldown-Menü genutzt, welches die für das aktuelle Merkmal entsprechende Funktion aufruft. Um unterschiedliche Jahre zu betrachten, wird ein Slider genutzt, welcher die Funktion des aktuellen Merkmals mit einem anderen Jahr aufruft.

4.1 Aufbereitung der Daten für GeoJSON

Hat man einen großen Datensatz, aus welchem man Daten in einer Karte visualisieren möchte, ist es zunächst notwendig, diese in das GeoJSON-Format zu überführen. Bei großen Datensätzen ist für eine solche Transformation die Nutzung eines Parsers und Speicherung der transformierten Daten in einer Datei sinnvoll, worauf hier nicht näher eingegangen wird.

Sollen die Daten beispielsweise in Form von, nach bestimmten Merkmalen eingefärbten, Polygonen angezeigt werden, so wird aus einer solchen GeoJSON-Datei etwa folgendes Objekt geladen:

```
var Stadtteile =
[
  {
    "type": "Feature",
    "properties": {
      "name" : "Stadtteil A",
      "Miete_2012": "250",
      "Miete_2013": "260",
      "Armut_2012": "15",
      "Armut_2013": "20"
    },
    "geometry": {
      "type": "Polygon",
```

```
      "coordinates": [[[12.44, 51.35], [12.45, 51.34], [12.42, 51.35],
          [12.44, 51.35]]]
    }
  },
  {
    "type": "Feature",
    "properties": {
      "name" : "Stadtteil B",
      "Miete_2012": "400",
      "Miete_2013": "420",
      "Armut_2012": "5",
      "Armut_2013": "6"
    },
    "geometry": {
      "type": "Polygon",
      "coordinates": [[[ 12.32, 51.36], [ 12.30, 51.35], [12.31, 51.36],
          [12.32, 51.36]]]
  }
}]
```

Jedes Merkmalsausprägung in einem bestimmten Jahr wird seperat in jedem Polygon gespeichert.

4.2 Das Pulldown-Menü

Um nun ein Merkmal auswählen zu können, bedienen wir uns eines Pulldown-Menüs. Das Pulldown-Menü befindet sich in der oberen rechten Ecke und hat die jeweiligen Merkmale zur Auswahl.

```
  var pulldownmenu = L.control({position: 'topright'});
  pulldownmenu.onAdd = function (map) {
    var div = L.DomUtil.create('div', 'pulldown');
    div.innerHTML = '<select><option id="miete">Miete</option>'
      .'<option id="armut">Armut</option></select>';
    div.firstChild.onmousedown = div.firstChild.ondblclick
      = L.DomEvent.stopPropagation;
    return div;
};
pulldownmenu.addTo(map);
```

Jedes Merkmal wird mit einer bestimmten `id` versehen, um besser darauf zugreifen zu können.

4.3 Der Jahresslider

Da in verschiedenen Jahren auch verschiedene Merkmalsausprägungen möglich sind, wird auf diese mit einem Slider zugegriffen, welcher sich unter der Karte befindet. In diesem Beispiel wird ein modifizierter Slider aus der jQueryUI-Bibliothek genutzt (die hierfür zu laden ist).

```
  $(document).ready(
  function(){
    var values = [ 2012, 2013];

    $("#slider").slider(
    {
      min: 2012,
```

```
8       max: values[values.length-1],
9       stepValues: values,
10      animate: "slow",
11      slide: function(event, ui)
12      {
13        var theSlider = $(this),
14        stepValues = theSlider.slider("option","stepValues"),
15        distance = [],
16        minDistance = theSlider.slider("option", "max"),
17        minI;
18
19        //Welcher erlaubte Wert ist der Position am naechsten?
20        //Hier wird einfach der kuerzeste Abstand gesucht
21        $.each(stepValues, function(i, val)
22        {
23          distance[i] = Math.abs(ui.value - val);
24          if(distance[i] < minDistance)
25          {
26            minDistance = distance[i];
27            minI = i;
28          }
29        });
30        //Wenn ein Wert gefunden wurde, wird dieser gesetzt
31        if(typeof minDistance === 'number' )
32        {
33          //Setzen des Wertes und schreiben des neuen Wertes in das 2. Div
34          theSlider.slider("value", stepValues[minI]);
35          $('#value').text(stepValues[minI]);
36          return false;
37        }
38        //Ansonsten stimmt irgendwas nicht und der aktuelle Wert wird
              beibehalten
39        else
40        {
41          //Der Wert wird nicht neu gesetzt, muss aber trotzdem in das Div
42          //geschrieben werden.
43          $('#value').text(ui.value);
44        }
45        return returnValue;
46      },
47      change: function( event, ui ) {
48        ...
49        //Die changefunktion wird in einem spaeteren Kapitel erklaert
50        ...
51      }
52    });
53  });
```

4.4 Die Stylefunktion

Die Stylefunktion regelt die Darstellung der Polygone. Hierbei wird pro Merkmal und Jahr eine eigene Stylefunktion verwendet. Beispielsweise sieht die Stylefunktion für die Miete im Jahr 2013 wie folgt aus:

```
1  function styleMiete_2013(feature) {
2    return {
```

```
    weight: 2,
    opacity: 1,
    color: 'white',
    dashArray: '3',
    fillOpacity: 0.7,
    fillColor: getColorMiete(feature.properties.Arbeitslose_2003)
  };
}
```

Diese ruft als Füllfarbe die Farbfunktion mit dem Wert des Merkmals aus der GeoJSON-Datei auf.

4.5 Die Farbfunktion

Anhand des Wertes der Ausprägung wird eine Farbe zurückgegeben. In welchen Abständen Ausprägungen zu einer gemeinsamen Farbdarstellung zusammengefasst werden sollten, ist je nach Datensatz natürlich sinnvoll zu wählen.

```
function getColorMiete(d) {
  return d > 400 ? '#810f7c' :
  d > 300  ? '#8856a7' :
  d > 200  ? '#8c96c6' :
  d > 100 ? '#b3cde3' :
  d > 0   ? '#edf8fb' :
  '#FFEDA0';
}
```

Die Funktion für die anderen Merkmale bzw. das andere Merkmal sieht dementsprechend ähnlich aus. In unserem Beispiel wäre diese `getColorArmut(d)`.

4.6 Legende

Um die Werte als Nutzer vernünftig interpretieren zu können ist es notwendig, eine Legende auf der Karte zu implementieren. Hierbei muss beachtet werden, dass meist für jedes Merkmal eine eigene Legende benötigt wird. Die Legende für die Miete sieht wie folgt aus:

```
var legendmiet = L.control({position: 'bottomright'});

legendmiet.onAdd = function (map) {
  aktuelleLegende = "legendmiet";

  var div1 = L.DomUtil.create('div1', 'info legend'),
  grades = [0, 100, 200, 300, 400],
  labels = [], from, to;

  for (var i = 0; i < grades.length; i++) {
    from = grades[i];
    to = grades[i + 1];
    labels.push(
      '<i style="background:' + getColorMiete(from + 1) + '"></i> ' +
      from  + (to ? '&ndash;' + to : '+'));
  }
  div1.innerHTML = labels.join('<br>');
  return div1;
};
```

Der Vektor `grades` muss hierbei die Werte beeinhalten, nach denen das Polygon eingefärbt wird. Die Variable `aktuelleLegende` beeinhaltet als Wert die gerade genutzte Legende. Dies ist notwendig, da bei Auswahl eines neuen Merkmals auch eine neue Legende verwendet werden muss. Dies ist in der Changefunktion des Pulldown-Menüs umgesetzt.

4.7 Changefunktion des Pulldown-Menüs

Hierbei wird zunächst das aktuelle Layer entfernt

```
$('select').change(function(){
    var id = $(this).children(":selected").attr("id");
    switch(aktuelleLegende){
      case "legendarmut":
        map.removeControl(legendarb);
        break;
      case "legendmiet":
        map.removeControl(legendmiet);
        break;
      default:
        //do nothing
        break;
    }
    change(id);
  });
```

In einer eigenen Funktion `change(id)` werden nun die Polygone neu eingefärbt und die neue Legende hinzugefügt.

```
function change(id) {
  map.removeLayer( geoJson ); //Entfernen des aktuellen Layers
  if (id == 'miete'){ //id aus dem Pulldownmenu
    legendmiet.addTo(map); //Neue Legende
    switch($("#slider").slider("value")){

      case 2012:
        geoJson = L.geoJson(states, {
          style: styleMiete_2012,
          onEachFeature: onEachFeature
        }).addTo(map);
        break;

      case 2013:
        geoJson = L.geoJson(states, {
          style: styleMiete_2012,
          onEachFeature: onEachFeature
        }).addTo(map);
        break;
    }
  }
  else if (id == 'armut'){
    legendarmut.addTo(map); //Neue Legende
    switch($("#slider").slider("value")){

      case 2012:
        geoJson = L.geoJson(states, {
          style: styleArmut2003,
          onEachFeature: onEachFeature
```

```
        }).addTo(map);
        break;

      case 2013:
        geoJson = L.geoJson(states, {
          style: styleArmut2003,
          onEachFeature: onEachFeature
        }).addTo(map);
        break;
    }
  }
}
```

Hierbei wird über ein `switch-case` auf den Wert des Sliders zugegriffen, je nach Jahr der entsprechende style für das GeoJSON-Layer festgelegt und dies zur Karte hinzugefügt.

4.8 Changefunktion des Sliders

Die Changefunktion des Sliders sieht ganz ähnlich aus, nur wird hier auf den Wert des Pulldown-Menüs zugegriffen.

```
change: function( event, ui ) {
  var pulldown = $('select').children(":selected").attr("id");
  map.removeLayer( geoJson );
  var slidevalue = ui.value;
  year = slidevalue;
  if (pulldown == 'miete'){

    switch(slidevalue){

      case 2012:
        geoJson = L.geoJson(states, {
          style: styleMiete_2012,
          onEachFeature: onEachFeature
        }).addTo(map);
        break;

      case 2013:
        geoJson = L.geoJson(states, {
          style: styleMiete_2013,
          onEachFeature: onEachFeature
        }).addTo(map);
        break;
    }
    else if (pulldown == 'armut'){

      switch(slidevalue){

        case 2012:
          geoJson = L.geoJson(states, {
            style: styleArmut2012,
            onEachFeature: onEachFeature
          }).addTo(map);
          break;

        case 2005:
          geoJson = L.geoJson(states, {
```

```
            style: styleMiete2005,
            onEachFeature: onEachFeature
          }).addTo(map);
          break;
      }
    }
}
```

4.9 Popups

Es ist immer sehr praktisch, die Polygone mit Popups zu versehen, welche die Merkmalsausprägung des jeweiligen Jahres enthalten.

```
function onEachFeature(feature, layer) {
  if(year == 2012){
    layer.bindPopup("<b>" + feature.properties.name +"</b><br>Miete:<i>"
      + feature.properties.Miete_2012 + "</i><br>Armut (in %):<i>"
      + feature.properties.Armut_2012 + "</i>");
  } else if (year == 2013){
    layer.bindPopup("<b>" + feature.properties.name +"</b><br>Miete:<i>"
      + feature.properties.Miete_2013 + "</i><br>Armut (in %):<i>"
      + feature.properties.Armut_2013 + "</i>");
  }
}
```

4.10 Elemente zur Karte hinzufügen

Wichtig ist es am Ende noch ein Layer mit default-Werten hinzuzufügen.

```
var geoJson;
geoJson = L.geoJson(states, {
  style: styleArbeitslose_2005,
  onEachFeature: onEachFeature
}).addTo(map);
```